МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА 44

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

старший преподаватель		А.В. Аксенов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине: Базы данных

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	B1441		Фурсов В.И.
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет № _____

Санкт-Петербург 2024

Тема курсовой работы:

Веб-приложение “Алкокалендарь”

Словесное описание области и актуальность:

В современном обществе наблюдается постоянный рост ассортимента алкогольной продукции, что приводит к увеличению ее потребления. В связи с этим возрастает важность проблемы контроля за употреблением алкоголя, учитывая его негативное влияние на здоровье. Разработка веб-приложения "Алкокалендарь" направлена на решение этой проблемы. Предполагаемые функции приложения включают в себя ведение календаря потребления алкоголя, а также учет затрат на его приобретение. Таким образом, данное приложение представляет собой инновационный инструмент для поддержания здоровья и благополучия, предоставляя пользователям эффективный способ отслеживания и управления своим потреблением алкоголя.

Описание данных, хранящихся в базе данных:

База данных должна содержать данные о:

- Пользователях, зарегистрировавшихся в системе и о добавленных им напитков.
- Алкогольной продукции: вид, цена, крепость, объем.
- Событиях: дата, место, напиток, потраченная сумма, количество выпитого.

Роли пользователей приложения:

- Человек

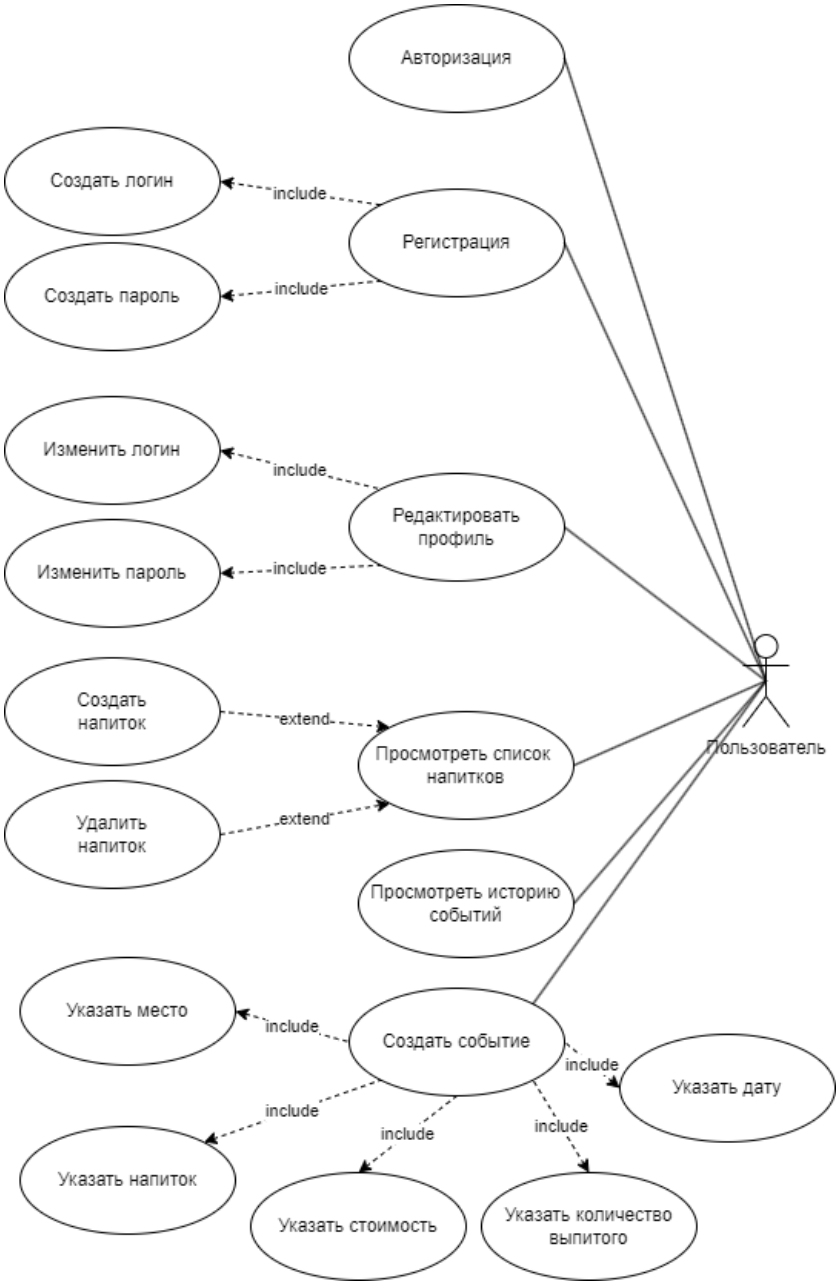
Развернутое описание функционала приложения для каждой из ролей:

- Человек

Пользователь может войти в систему, используя свою учетную запись, либо зарегистрироваться, если учетной записи еще нет. После авторизации пользователю доступен профиль, где он может изменить информацию о себе, такую как логин и пароль. В профиле ему доступна статистика о суммарном количестве выпитого алкоголя и потраченных денег.

Пользователь может создавать события, когда он употреблял алкогольные напитки. Вводимая информация включает в себя: Дата события, локация, вид напитка, его крепость, стоимость, объем, а также причины употребления, описанные в заметках. Пользователю доступна история всех дней, когда были употреблены алкогольные напитки.

Диаграмма вариантов использования:



Предполагаемые технологии и платформа реализации:

- СУБД: SQLite
- ОС: Windows
- Язык программирования: Python
- Фреймворк: Flask
- Тип приложения: Веб-Приложение

Срок представления курсовой работы:

07.02.2024

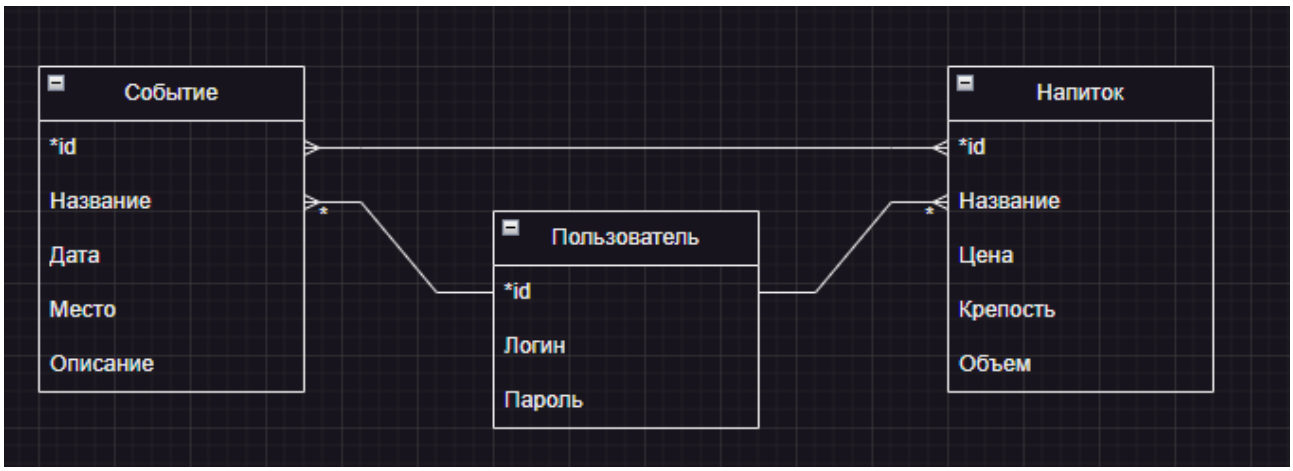


Рисунок 1 — Диаграмма «Сущность-связь»

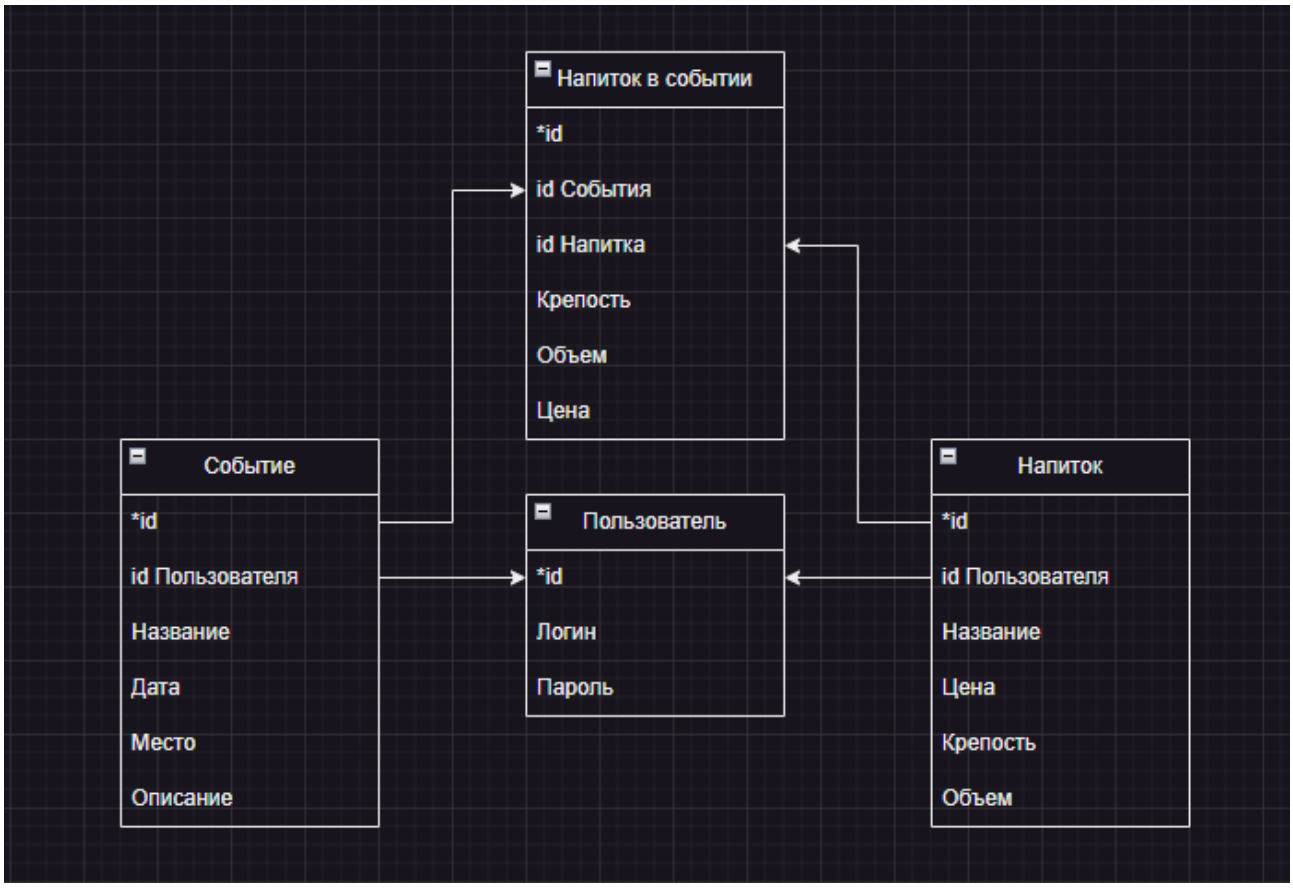


Рисунок 2 — Реляционная диаграмма

Для выполнения темы курсовой работы были использованы следующие инструменты: Flask и SQLite. Flask – это фреймворк для создания веб-приложений на Python, относящийся к категории микрофреймворков. Он применяется для реализации логики на веб-сайте и взаимодействия с базой данных. SQLite – компактная встраиваемая система управления базами данных. Взаимодействие с ней осуществляется через Python.

Для создания базы данных используется выполнение SQL-кода. При запуске веб-приложения необходимо установить соединение с созданной базой данных для последующего взаимодействия с ней. Затем в программе база данных объявляется с использованием класса, который содержит текущую базу данных и ее курсор. В Python для предотвращения возможных ошибок взаимодействие с базой данных происходит в блоке try-except. Используемые методы включают: cursor.execute() для выполнения SQL-кода из строки; db.commit() для сохранения изменений в базе данных; cursor.fetchone/fetchall() для получения упорядоченного списка из курсора одного/всех выбранных элементов.

Приложение запускается путем выполнения файла main.py

Файл main.py. В нем инициализируется база данных, а также в нем реализованы переходы по страницам и логика приложения.

Текст программы:

```
from data import Data
from flask import Flask, render_template, url_for, request, g,
redirect, flash
from flask_login import LoginManager, login_user, login_required,
logout_user, current_user
from werkzeug.security import generate_password_hash,
check_password_hash
from user import User
import sqlite3
import os

DATABASE = "/db/database.db"
```

```
DEBUG = True
SECRET_KEY = "12345"

app = Flask(__name__)
app.config.from_object(__name__)
app.config.update(dict(DATABASE=os.path.join(app.root_path,
"database.db")))

login_manager = LoginManager(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    print("load user")
    return User().fromDB(user_id, D)

def connect_db():
    connect = sqlite3.connect(app.config['DATABASE'])
    connect.row_factory = sqlite3.Row
    return connect

def create_db():
    print("ПОДКЛЮЧЕНИЕ")
    db = connect_db()
    with app.open_resource("database.sql", mode= "r") as script:
        db.cursor().executescript(script.read())
    db.commit()
    db.close()

def get_db():
    if not hasattr(g, "link_db"):
        g.link_db = connect_db()
        print("БАЗА ДАННЫХ ПОЛУЧЕНА")
    return g.link_db

D = None
@app.before_request
def before_request():
    global D
    db = get_db()
    D = Data(db)


@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'link_db'):
        g.link_db.close()

@app.route("/")
@app.route("/index")
def index():
    return render_template("index.html")
```

```
@app.route("/settings", methods = ["GET", "POST"])
def settings():
    if request.method == "POST":
        if "new_name" in request.form:
            D.update_name(current_user.get_id(),
request.form["new_name"])
            flash("Вы сменили имя")
            return redirect(url_for('settings'))

        if "new_password" in request.form:
            D.update_password(current_user.get_id(),
request.form["new_password"])
            flash("Вы сменили пароль")
            return redirect(url_for('settings'))

    return render_template("settings.html", name =
current_user.get_nickname())

@app.route("/profile")
@login_required
def profile():
    print("starter drinks: ", current_user.starter_drinks)
    return render_template("profile.html",
                           nickname = current_user.get_nickname(),
                           id = current_user.get_id(),
                           fauvorite_drink =
D.get_fauvorite_drink(current_user.get_id()),
                           stats =
D.get_statistic(current_user.get_id())
                           )

@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect("/login")

@app.route("/create_event", methods = ["POST", "GET"])
@login_required
def create_event():
    drinks = D.get_drinks(current_user.get_id())

    if request.method == "POST":
        D.add_event(current_user.get_id(),
                    request.form["title"],
                    request.form["event_date"],
                    request.form["place"],
                    request.form["description"])
        flash("Вы успешно добавили событие")
        return redirect(url_for("history"))
    return render_template("create_event.html", drinks = drinks)
```

```
@app.route("/drinks", methods = ["POST", "GET"])
@login_required
def drinks():
    drinks = D.get_drinks(current_user.get_id())

    if request.method == "POST":
        D.add_drink(request.form["title"],
                    request.form["price"],
                    request.form["alcohol"],
                    request.form["volume"],
                    current_user.get_id())

        flash("Вы успешно добавили напиток.")

        return redirect(url_for("drinks"))


    return render_template("drinks.html", drinks = drinks)

@app.route("/delete_drink/<int:drink_id>")
def delete_drink(drink_id):
    D.delete_drink(drink_id)
    flash("Напиток удален")
    return redirect(url_for('drinks'))


@app.route("/history")
@login_required
def history():
    events = D.get_events(current_user.get_id())
    print(events)
    return render_template("history.html", events = events)

@app.route("/event/<int:event_id>", methods = ["POST", "GET"])
@login_required
def event(event_id):
    if request.method == "POST":
        D.add_drink_in_event(current_user.get_id(), event_id,
request.form["drink"], request.form["volume"],
request.form["price"])
        flash("Напиток добавлен", "accept")
        return redirect(url_for('event', event_id = event_id))
    return render_template("event.html",
                           drinks =
D.get_drinks(current_user.get_id()),
                           drinks_in_event =
D.get_drinks_in_event(event_id),
                           volume_sum =
D.get_sum_of_volume(event_id),
                           price_sum =
D.get_sum_of_price(event_id),
```

```
                            description =
D.get_description(event_id),
                            event_info = D.get_event_info(event_id))

@app.route("/delete_event/<int:event_id>")
def delete_event(event_id):
    D.delete_event(event_id)
    flash("Событие удалено")
    return redirect(url_for('history'))

@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        user = D.get_user_by_nickname(request.form['nickname'])
        if user and check_password_hash(user['psw'],
request.form['psw']):
            user_login = User().create(user)
            login_user(user_login)

            return redirect("/")
    return render_template("login.html")

@app.route("/register", methods = ["POST", "GET"])
def register():
    if request.method == "POST":
        hash = generate_password_hash(request.form["psw"])
        D.add_user(request.form['nickname'], hash)
        flash("Успешная регистрация", "accept")
        return redirect("/login")
    return render_template("register.html")


if __name__ == "__main__":
    create_db()
    app.run(debug=True)
```

Файл data.py. В нем реализован один класс, методы которого представляют из себя запросы к базе данных с получением или внесением информации.

Текст программы:

```
from flask import Flask, render_template, url_for, request,
redirect, flash
from werkzeug.security import generate_password_hash
from sqlite3 import Error as error

class Data:
    def __init__(self, database):
        self.db = database
        self.cursor = database.cursor()
```

```
    def add_drink(self, title, price, alcohol, volume, user_id):
        try:
            self.cursor.execute(f"INSERT INTO drink
VALUES(NULL, ?, ?, ?, ?, ?)", (title, price, alcohol, volume,
user_id))
            self.db.commit()
        except error:
            print("ОШИБКА! " + str(error))

    def get_drinks(self, user_id):
        try:
            self.cursor.execute(f"SELECT id, title, alcohol, volume,
price FROM drink WHERE user_id = {user_id} ORDER BY title")
            drinks = self.cursor.fetchall()

            return drinks
        except error:
            print("ОШИБКА! " + str(error))

    def get_drink_by_id(self, drink_id):
        try:
            self.cursor.execute(f"SELECT title FROM drink WHERE id =
{drink_id}")

        except error:
            print("ОШИБКА! " + str(error))

    def get_fauvorite_drink(self, user_id):
        try:
            self.cursor.execute(f"""SELECT drink_title
                                    FROM events_drink
                                    WHERE user_id = {user_id}
                                    GROUP BY drink_title
                                    ORDER BY COUNT(*) DESC
                                    LIMIT 1;""")

            fauvorite_drink = self.cursor.fetchone()

            return fauvorite_drink

        except error:
            print("ОШИБКА! " + str(error))

    def delete_drink(self, drink_id):
        try:
            self.cursor.execute(f"DELETE FROM drink WHERE id =
{drink_id}")
            self.db.commit()

            return redirect(url_for('drinks'))
```

```
        except error:
            print("ОШИБКА! " + str(error))

    def add_user(self, nickname, hash):
            try:
                self.cursor.execute(f"SELECT COUNT() as 'count' FROM
user WHERE nickname LIKE '{nickname}' ")
                res = self.cursor.fetchone()

                if res['count'] > 0:
                    flash("ПОЛЬЗОВАТЕЛЬ С ТАКИМ ИМЕНЕМ УЖЕ
СУЩЕСТВУЕТ")
                    print("ПОЛЬЗОВАТЕЛЬ С ТАКИМ ИМЕНЕМ УЖЕ
СУЩЕСТВУЕТ")

                self.cursor.execute("INSERT INTO user
VALUES(NULL, ?, ?)", (nickname, hash))
                self.db.commit()

            except error:
                print("ОШИБКА! " + str(error))

    def get_user_by_nickname(self, nickname):
        try:
            self.cursor.execute(f"SELECT * FROM user WHERE nickname
= '{nickname}' LIMIT 1")
            res = self.cursor.fetchone()

            return res
        except error:
                print("ОШИБКА! " + str(error))

    def get_user(self, user_id):
        try:
            self.cursor.execute(f"SELECT * FROM user WHERE id =
{user_id} LIMIT 1")
            user = self.cursor.fetchone()
            return user

        except error:
                print("ОШИБКА! " + str(error))

    def update_name(self, user_id, new_name):
        try:
            self.cursor.execute(f"UPDATE user SET nickname =
'{new_name}' WHERE id = {user_id} ")
            self.db.commit()

        except error:
                print("ОШИБКА! " + str(error))

    def update_password(self, user_id, new_password):
```

```
        try:
            psw = generate_password_hash(new_password)
            self.cursor.execute(f"UPDATE user SET psw = '{psw}'
WHERE id = {user_id} ")
            self.db.commit()

        except error:
                print("ОШИБКА! " + str(error))

    def get_statistic(self, user_id):
        try:
            self.cursor.execute(f"SELECT SUM(price), SUM(volume)
FROM events_drink WHERE user_id = {user_id}")
            stats = self.cursor.fetchone()
            return stats

        except error:
                print("ОШИБКА! " + str(error))


    def add_event(self, user_id, title, event_date, place,
description):
        try:
            self.cursor.execute(f"INSERT INTO events VALUES(NULL, ?,
?, ?, ?, ?)", (user_id, title, event_date, place, description))
            self.db.commit()
        except error:
            print("ОШИБКА! " + str(error))


    def add_drink_in_event(self, user_id, event_id, drink, volume,
price):
        try:
            self.cursor.execute(f"SELECT id, alcohol FROM drink
WHERE title = '{drink}' AND user_id = {user_id}")
            drink_id = self.cursor.fetchone()

            self.cursor.execute(f"INSERT INTO events_drink
VALUES(NULL, ?, ?, ?, ?, ?, ?, ?)", (user_id, event_id, drink_id[0],
drink, drink_id[1], volume, price))
            self.db.commit()
        except error:
            print("ОШИБКА! " + str(error))

    def get_drinks_in_event(self, event_id):
        try:
            self.cursor.execute(f"SELECT drink_id, drink_title,
drink_alcohol, volume, price FROM events_drink WHERE event_id =
{event_id}")
            drinks_in_event = self.cursor.fetchall()
            return drinks_in_event
        except error:
```

```
            print("ОШИБКА! " + str(error))

    def get_sum_of_volume(self, event_id):
        try:
            self.cursor.execute(f"SELECT SUM(volume) FROM
events_drink WHERE event_id = {event_id}")
            volume_sum = self.cursor.fetchone()
            return volume_sum
        except error:
            print("ОШИБКА! " + str(error))

    def get_sum_of_price(self, event_id):
        try:
            self.cursor.execute(f"SELECT SUM(price) FROM
events_drink WHERE event_id = {event_id}")
            price_sum = self.cursor.fetchone()
            return price_sum
        except error:
            print("ОШИБКА! " + str(error))


    def get_events(self, user_id):
        try:
            self.cursor.execute(f"SELECT id, title, event_date,
place, descript FROM events WHERE user_id = {user_id} ORDER BY
event_date DESC")
            events = self.cursor.fetchall()
            return events

        except error:
                print("ОШИБКА! " + str(error))

    def get_description(self, event_id):
        try:
            self.cursor.execute(f"SELECT descript FROM events WHERE
id = {event_id}")
            description = self.cursor.fetchone()
            return description
        except error:
                print("ОШИБКА! " + str(error))

    def get_event_info(self, event_id):
        try:
            self.cursor.execute(f"SELECT event_date, title, place,
descript FROM events WHERE id = {event_id}")
            event_info = self.cursor.fetchone()

            return event_info
        except error:
                print("ОШИБКА!" + str(error))

    def delete_event(self, event_id):
```

```
        try:
            self.cursor.execute(f"DELETE FROM events_drink WHERE
event_id = {event_id}")
            self.cursor.execute(f"DELETE FROM events WHERE id =
{event_id}")
            self.db.commit()

            return redirect(url_for('history'))

        except error:
            print("ОШИБКА! " + str(error))
```

Файл user.py. Нужен для использования модуля flask-login. Состоит из одного класса, позволяет взаимодействовать с авторизованным пользователем.

Текст программы:

```
class User():
    starter_drinks = False

    def fromDB(self, user_id, db):
        self.__user = db.get_user(user_id)
        return self

    def create(self, user):
        self.__user = user
        return self

    def is_authenticated(self):
        return True

    def is_active(self):
        return True

    def is_anonymous(self):
        return False

    def get_id(self):
        return str(self.__user['id'])

    def get_nickname(self):
        return str(self.__user["nickname"])
```

файл database.sql. Нужен для создания базы данных.

Текст программы:

```
CREATE TABLE IF NOT EXISTS user
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nickname TEXT NOT NULL UNIQUE,
    psw TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS drink
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    price INTEGER NOT NULL,
    alcohol INTEGER NOT NULL,
    volume REAL NOT NULL,
    user_id INTEGER,

    FOREIGN KEY(user_id) REFERENCES user(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS events
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    title TEXT,
    event_date DATE NOT NULL,
    place TEXT,
    descript TEXT,

    FOREIGN KEY(user_id) REFERENCES user(id)
);

CREATE TABLE IF NOT EXISTS events_drink
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    event_id INTEGER,
    drink_id INTEGER,
    drink_title TEXT,
    drink_alcohol INTEGER,
    volume INTEGER,
    price INTEGER,

    FOREIGN KEY(user_id) REFERENCES user(id) ON DELETE CASCADE,
    FOREIGN KEY(event_id) REFERENCES events(id) ON DELETE CASCADE,
    FOREIGN KEY(drink_id) REFERENCES drink(id),
    FOREIGN KEY(drink_alcohol) REFERENCES drink(alcohol)
);
```

Описание сценариев использования приложения

На главной странице нет особого функционала. Но на всех страницах присутствует боковая панель, которая используется для навигации по приложению.

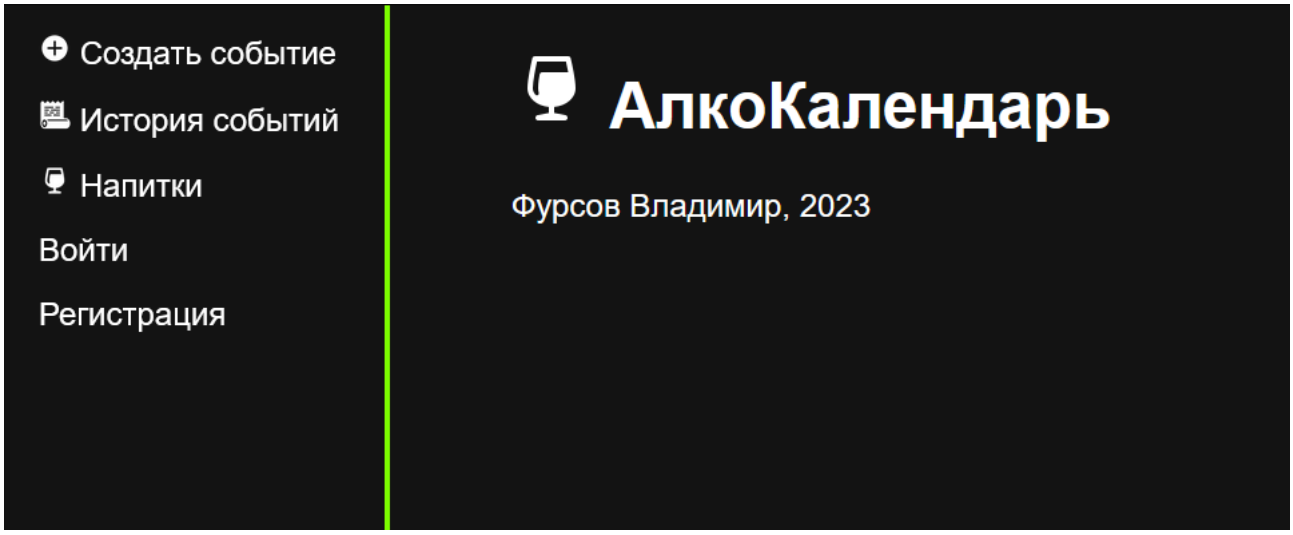


Рисунок 3 — Главная страница

Окна регистрации и авторизации.

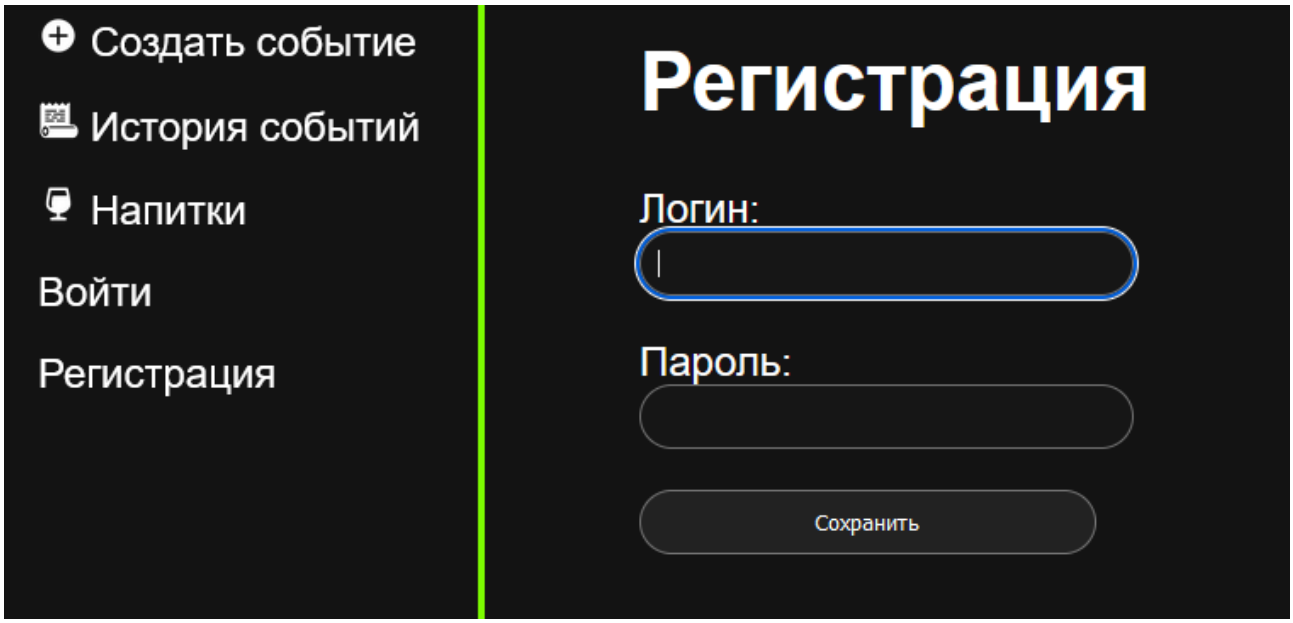


Рисунок 4 — Регистрация

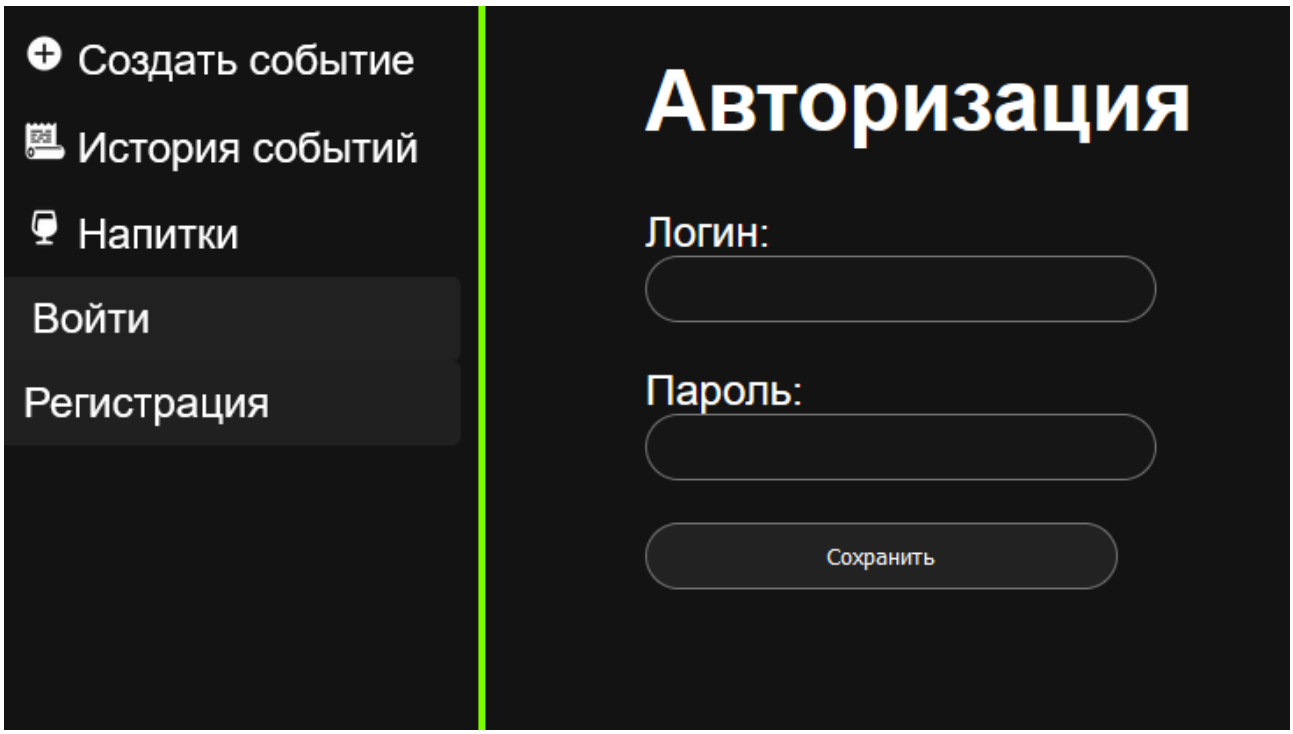


Рисунок 5 — Авторизация

После авторизации пользователю становятся доступны: его профиль, возможность создания и просмотра событий, меню напитков и настройки.

Для того, чтобы полноценно пользоваться приложением, необходимо создать напиток. В меню создания напитков можно выбрать его название, крепость, объем в литрах и цену за бутылку. Ниже отображается список напитков пользователя.

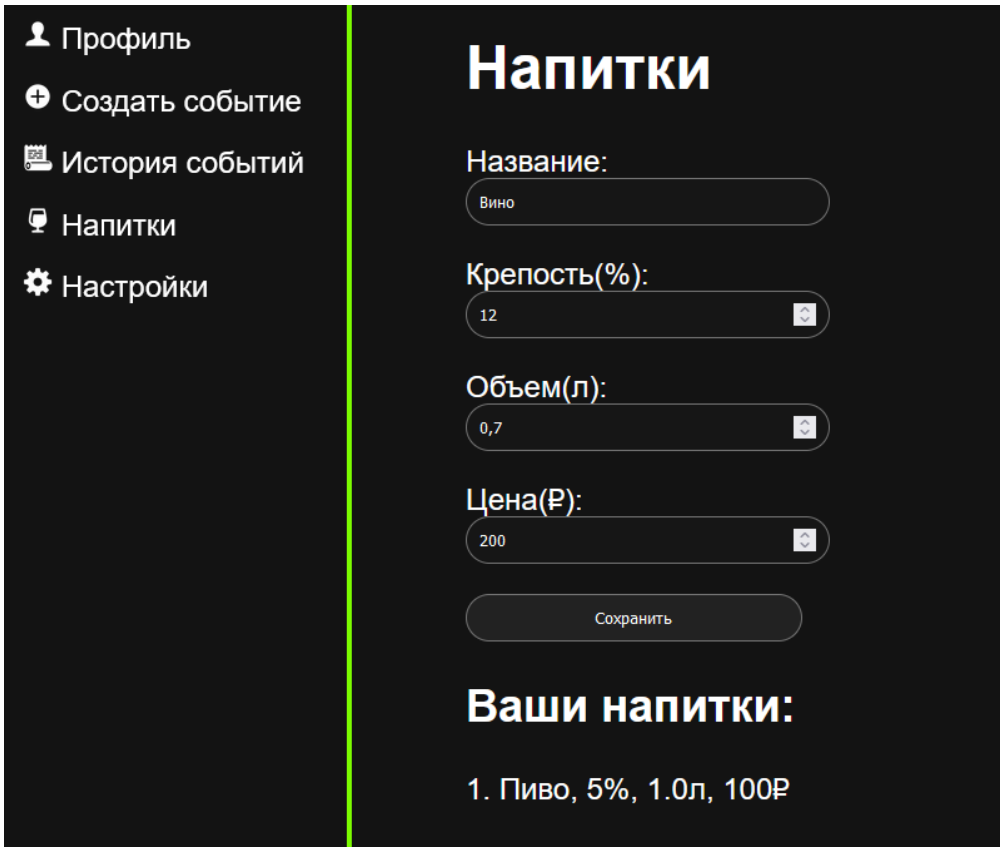


Рисунок 6 — Меню напитков

Теперь можно создать событие. Событие представляет из себя название, дату, место проведения и описание.

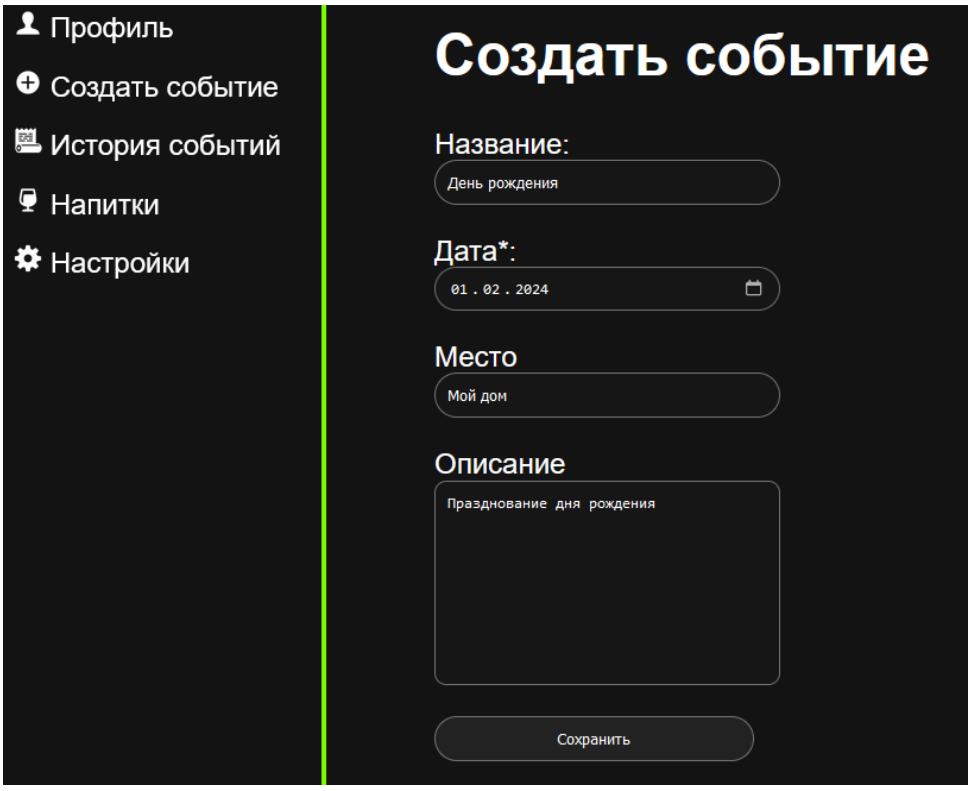


Рисунок 7 — Меню создания событий

После этого пользователю становится доступна история событий.

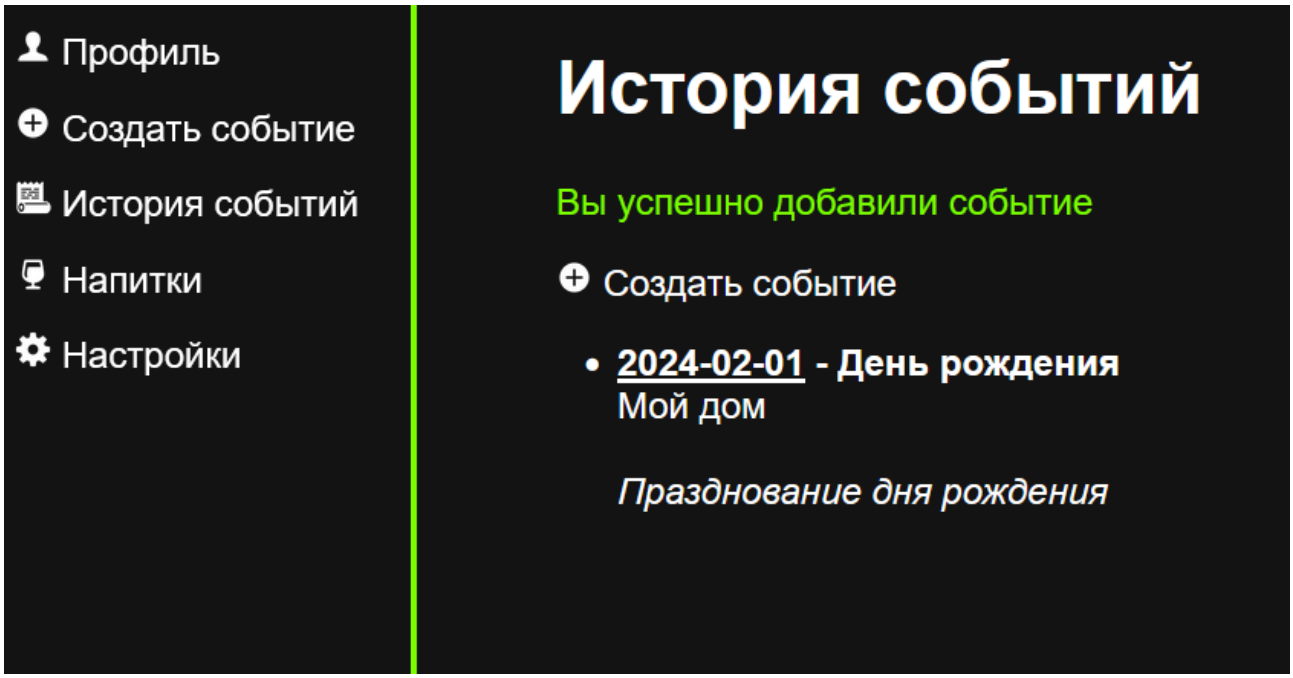


Рисунок 8 — История событий

Каждое событие можно открыть и перейти в меню управление событием.

В нем помимо основной информации отображается список выпитых напитков и статистика за это событие. Ниже можно добавить выпитый напиток, выбрав из списка напитков пользователя и добавив количество в миллилитрах и стоимость.

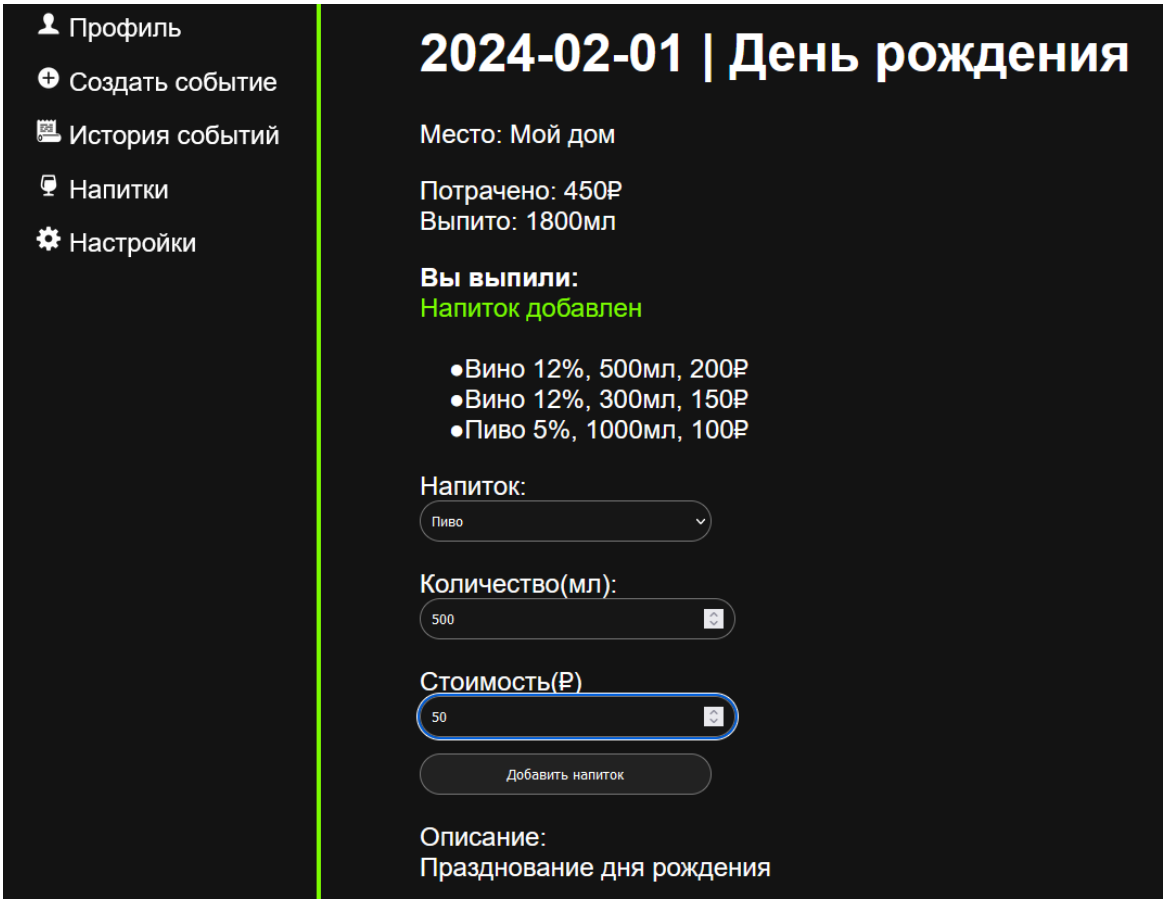


Рисунок 9 — Меню управления событием

В окне профиля пользователь видит статистику со всех событий: количество выпитого, потраченных денег и наиболее часто выпиваемый напиток. Также пользователь может выйти из профиля.

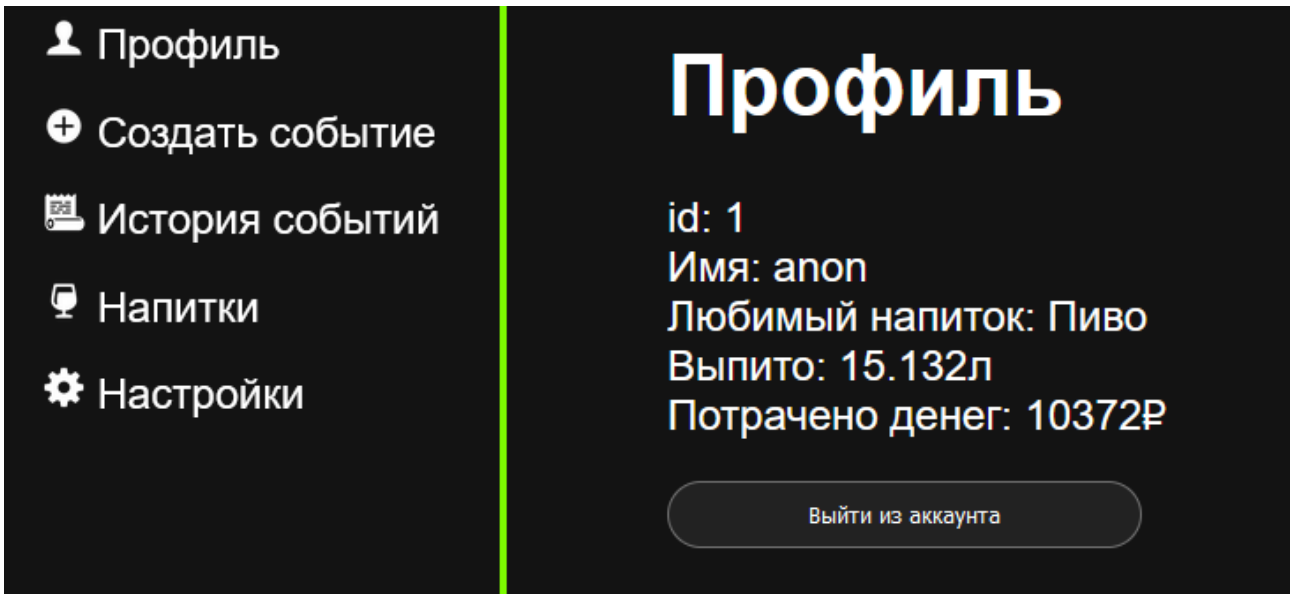


Рисунок 10 — Окно профиля

В окне настроек пользователь может сменить имя и пароль.

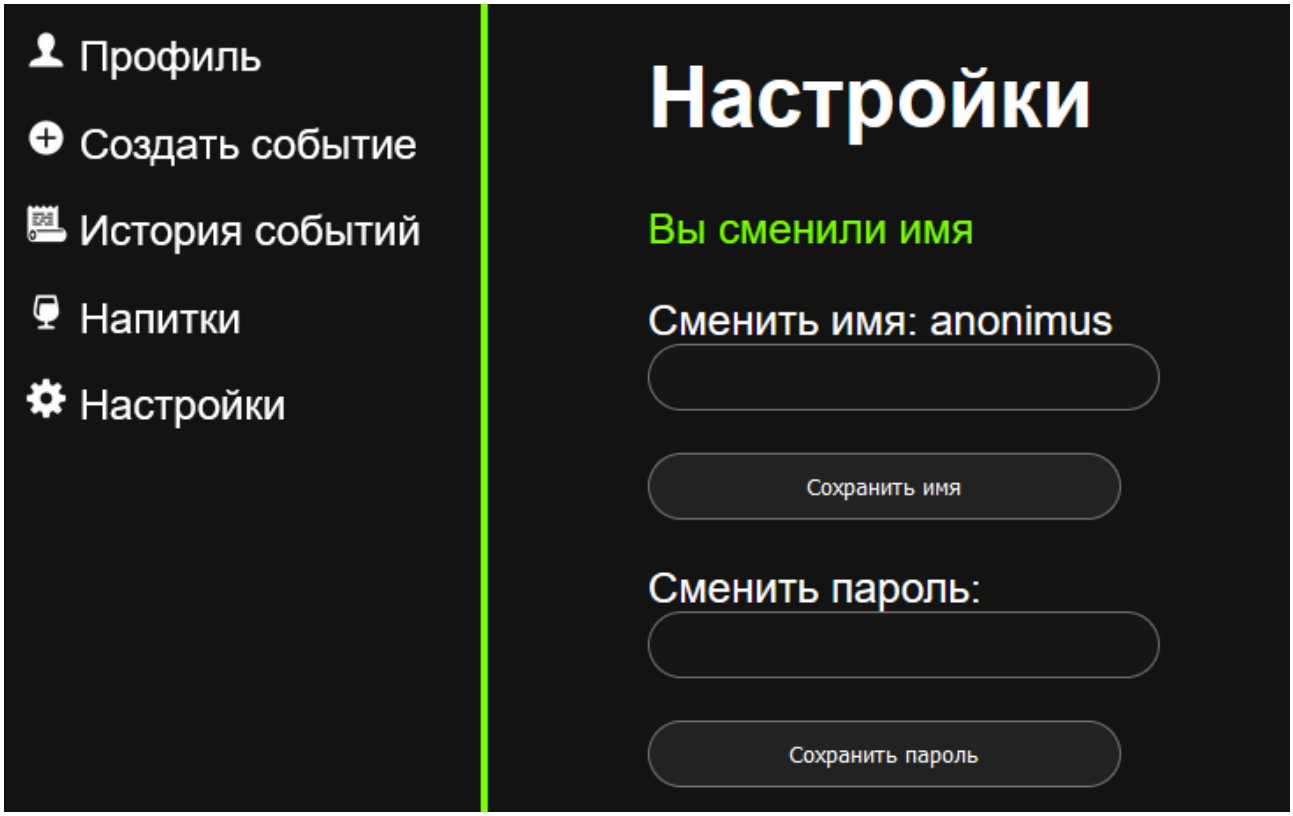


Рисунок 11 — Меню настроек

Вывод: В результате выполнения лабораторной работы было создано веб-приложение на языке Python с использованием фреймворка Flask и СУБД SQLite по теме «Алкокалендарь».